The 38th IEEE Conference on Decision and Control, CDC'99, Phoenix, Arizona, USA, Dec 7-10, 1999

Modelica Hybrid Modeling and Efficient Simulation

Sven Erik Mattsson
Dynasim AB
Research Park Ideon
SE-223 70 Lund, Sweden
SvenErik@Dynasim.se

Martin Otter
DLR Oberpfaffenhofen
D-82230 Wessling
Germany
Martin.Otter@DLR.de

Hilding Elmqvist
Dynasim AB
Research Park Ideon
SE-223 70 Lund, Sweden
Elmqvist@Dynasim.se

Abstract

Modelica is an object-oriented language for modeling of large and heterogeneous physical systems. Typical applications include mechatronic models in automotive and aerospace applications involving mechanical, electrical and hydraulic subsystems as well as control systems.

Modeling of an ideal diode and Coulomb friction is discussed to illustrate the unique hybrid features of Modelica. The language has been designed to allow tools to generate efficient code automatically. Approaches supported by the Dynamic Modeling Laboratory Dymola from Dynasim are presented. Real-time simulation of an automatic gearbox is discussed to demonstrate the power of symbolic manipulation. A gearbox is inherently hybrid, since the structure varies during each gearshift. Friction is also an important phenomenon. It takes Dymola only a few seconds to translate a Modelica model of an automatic gearbox with 11 switching elements into efficient simulation code. A 500 MHz DEC alpha processor from dSPACE evaluates one Euler step including a possible mode switch in less than 0.18 ms.

Introduction

Modelica™ is a uniform object-oriented language for modeling of physical systems. It is a modern language built on *non-causal* modeling with *mathematical equations* and *object-oriented* constructs to facilitate reuse of modeling knowledge in order to support effective library development and model exchange. The features of Modelica to model continuous-time systems described by differential-algebraic equations, DAEs, are discussed in [4, 6, 10]. The hybrid features are discussed in [11]. For details about the Modelica project, see http://www.Modelica.org/.

Here the focus is on the hybrid features to model variable structure systems and on approaches for efficient simulation. Modeling of an ideal diode and Coulomb friction and simulation of systems involving such components are illustrated. Real-time simulation of automatic gearboxes is discussed to demonstrate the power of symbolic manipulation.

Hybrid Modeling in Modelica

For continuous-time systems, object-oriented modeling languages like Dymola, gPROMS, Modelica and Omola, are based on the same principle: using DAEs to mathematically describe model components. For discrete event systems this is different, because there does not exist a single widely accepted description form. Instead, many formalisms are available, e.g., finite automata, Petri nets, statecharts, sequential function charts, DEVS, logical circuits, difference equations, CSP, process-oriented languages that are all suited for particular application areas.

In Modelica the central property is the usage of *synchronous* differential, algebraic and discrete equations [11]. The idea of using the synchronous data flow principle in the context of hybrid systems was introduced in [5]. For pure *discrete event* systems, the same principle is utilized in synchronous languages [8] such as SattLine [3], Lustre [9] and Signal [7], in order to arrive at safe implementations of real-time systems and for verification purposes.

If a physical component is modeled detailed enough, there are usually no discontinuities in the system. When neglecting some "fast" dynamics, in order to reduce simulation time and modeling effort, discontinuities appear in a physical model. As a typical example, consider the diode shown in Figure 1. If the detailed switching behavior is neglectable with regards to other modeling effects, it is often sufficient to use the ideal diode characteristic, which typically gives a simulation speedup of 1 to 2 order of magnitudes.

It is straightforward to model the real diode characteristic in the left part of Figure 1, because the

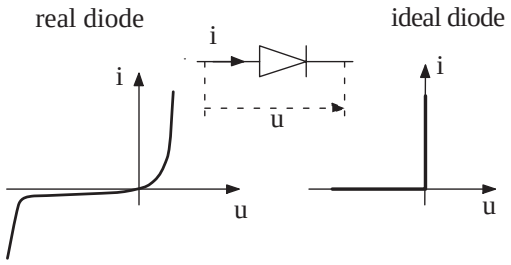
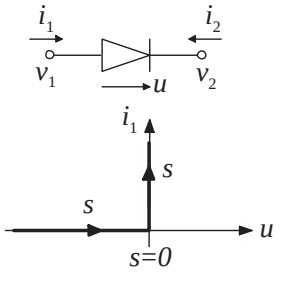


Figure 1: Real and ideal diode characteristics.

current i has just to be given as an analytic or a tabulated function of the voltage drop u. It is more difficult to model the ideal diode characteristic in the right part of Figure 1, because the current at $u = 0$ is no longer a function of u, i.e., a mathematical description in the form $i = i(u)$ is no longer possible. This problem can be solved by recognizing that a curve can also be described in a parameterized form $u = u(s)$ and $i = i(s)$ by introducing a curve parameter s. This description form is more general and allows us to describe an ideal diode *uniquely* in a *declarative* way. For the ideal diode we get the equations shown in Figure 2. The set of equations is easily understandable, in particular if it comes with a plot of the ideal diode characterstic with the indicated s-parameterization. Moreover, it a unique, mathematical description of the ideal diode and contains no algorithm. It is possible to write the ideal diode in a declarative form without using the idea of s-parameterization: $0 =$ **if** $u > 0$ **or** $i > 0$ **then** v **else** i. However, it is more difficult to understand.

A s-parameter model of an ideal GTO thyristor is similar to the diode model, but uses the equation "*off* $= s < 0$ **or not** *fire*" and for an ideal thyristor use "*off* $= s < 0$ **or pre**(*off*) **and not** *fire*", where **pre**(x) is the left limit, $x(t^-)$ of a variable x at time t.

The technique of parameterized curve descriptions was introduced in [1] and a series of related papers. However, no proposal was yet given how to actually implement such models in a numerically sound way. In Modelica the (new) solution method follows logically because the equation based system naturally leads to a system of mixed continuous/discrete equations which have to be solved at event instants.



$$
\begin{aligned}
0 &= i_1 + i_2 \\
u &= v_1 - v_2 \\
\mathit{off} &= s < 0 \\
u &= \textbf{if } \mathit{off} \textbf{ then } s \textbf{ else } 0 \\
i_1 &= \textbf{if } \mathit{off} \textbf{ then } 0 \textbf{ else } s
\end{aligned}
$$

Figure 2: Ideal diode model

Efficient Simulation

Approaches to efficient simulation of hybrid models will now be discussed. These are supported by the Dynamic Modeling Laboratory, Dymola [2].

In order to get a better understanding, let us discuss simulation of the simple rectifier circuit in Figure 3. Assume that the generator voltage, $v_0(t)$, is a known function of time. Collecting the equations of all components and connections, eliminating trivial equations of the forms $a = b$ and $a = -b$ and sorting (assuming the states, here v_2, to be known), gives

$$
\begin{aligned}
\mathit{off} &= s < 0 \\
u &= \textbf{if } \mathit{off} \textbf{ then } s \textbf{ else } 0 \\
i_0 &= \textbf{if } \mathit{off} \textbf{ then } 0 \textbf{ else } s \\
u &= v_1 - v_2 \\
R_1 \cdot i_0 &= v_0(t) - v_1
\end{aligned} \qquad (1)
$$

$$
\begin{aligned}
i_2 &:= v_2 / R_2 \\
i_1 &:= i_0 - i_2 \\
\tfrac{dv_2}{dt} &:= i_1 / C
\end{aligned}
$$

The first 5 equations are coupled and build a system of equations in the 5 unknowns: *off*, s, u, v_1 and i_0. The remaining assignment statements leads to the state derivative $\dot{v}_2$.

Continuous simulation and event detection

During continuous integration the Boolean variables, i.e., *off*, are fixed and the Boolean equations (1.1) are not evaluated. However, to be able to detect when Boolean values are to be changed, all relations involving continuous time variables are extracted and converted to crossing functions (here s) which the solver monitors during continuous integration. When any crosses zero, the solver calculates the crossing time and halts the integration. This is the standard approach to handle discontinuities properly.

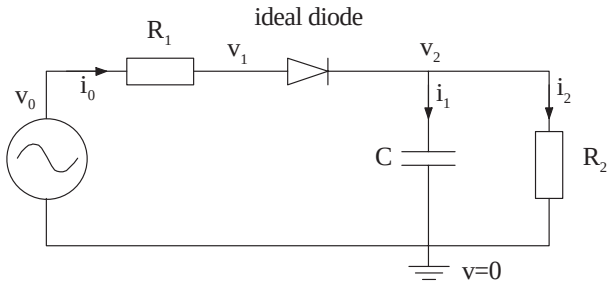


Figure 3: Simple rectifier circuit.

Tearing

Algebraic loops can be solved in various ways during the continuous integration. It is advantageous to reduce the size of the problem sent to a numerical solver. It is easy to solve for the variables u, i_0 and v_1 from (1.2-1.4) and calculate a residual for the remaining equation (1.5), when s is known. A numerical solver need only to consider the unknown s. This approach is called *tearing*. Let z represent the unknowns to be solved from the equation system; the components of z include unknown derivatives and unknown algebraic variables. The idea of tearing is to partition z into two parts z_1 and z_2 such that

$$\begin{aligned} Lz_1 &= f_1(z_2) \\ 0 &= f_2(z_1, z_2) \end{aligned}$$

where L is lower triangular with constant, non-zero diagonal. A numerical solver needs then only consider z_2 as unknown, because it is simple to calculate the residual $\delta = f_2(L^{-1}f_1(z_2), z_2)$ when z_2 is given. The aim is of course to make the number of components of z_2 as small as possible. It is an NP-complete problem to find the minimum. However, there are fast heuristic approaches to find good partitionings of z.

If the equations are linear, they can be written as

$$\begin{aligned} Lz_1 &= Az_2 + b_1 \\ 0 &= Bz_1 + Cz_2 + b_2 \end{aligned}$$

and it is possible to eliminate z_1 symbolically to get

$$Jz_2 = b$$

where

$$\begin{aligned} J &= C + BL^{-1}A \\ b &= b_2 + BL^{-1}b_1 \end{aligned}$$

This may be interpreted as Gauss elimination of z_1.

Individual simulation code for modes

Algebraic loops are inherent for variable structure systems. For a specific mode, these may reduce or fall a part because its has more structural zeros.

Reconsider the equation system (1.2-1.5). Inspection shows that the structure of the problem depends critically on the value of *off*. If *off* is fixed to **false** or **true**, the algebraic loop degenerates and it is trivial to solve the equation system symbolically into a sequence of assignments:

if not *off* **then**	**if** *off* **then**
$u := 0$	$i_0 := 0$
$v_1 := v_2$	$v_1 := v_0(t)$
$i_0 := (v_0(t) - v_1)/R$	$u := v_1 - v_2$
$s := i_0$	$s := u$

Mixed equation systems

At an *event instant*, the equations (1.1-1.5) constitute a mixed system having Real and *Boolean* unknowns. Dymola solves such problems provided that for each Boolean unknown there is an equation which in fact is an assignment for it. Dymola solves the real unknowns in usual ways and fix point iterates over the Boolean unknowns. Our experiences of the approach is good. The alternative of substituting the Boolean variables to get a continuous non-linear problem has bad numerical properties.

Code optimization

To improve efficiency, Dymola pre-evaluates expressions and sectionizes code to prevent multiple evaluations of expressions that are constant during a simulation run and to minimize the number of expressions to evaluate during continuous time integration. Dymola also searches for common subexpression and generates code for analytic Jacobians.

Friction

The simulation of components with ideal switch elements becomes difficult, if switching results in an index change of the DAE, i.e., if the number of states is changing. A typical example is Coulomb friction where this situation is present even in the most simple case. To concentrate on the essentials, first the simplified friction element in Figure 4 is discussed.

The friction force f acts between two surfaces, see right part of Figure 4, and is a linear function of the relative velocity v between the friction surfaces when the surfaces are sliding relative to each other. When the relative velocity becomes zero, the two surfaces are stuck to each other and the friction force is no longer a function of v. The element starts sliding again if the friction force becomes larger than the maximum static friction force f_0. This element can also be described as a parameterized curve

```
forward  = s >  1;   backward = s < -1;
v=if forward  then  s - 1       else
  if backward then  s + 1       else 0;
f=if forward  then  f0+f1*(s-1) else
  if backward then -f0+f1*(s+1) else f0*s;
```

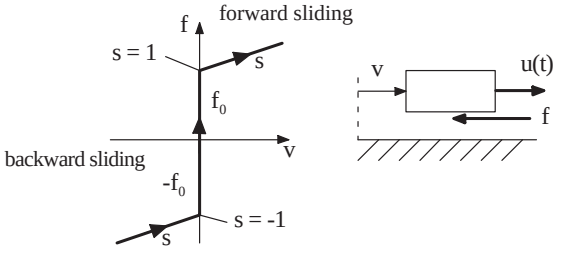


Figure 4: Simplified Coulomb friction element.

This model completely describes the simplified friction element in a *declarative* way. Unfortunately, we have not so far developed a method to transform automatically such an element description *automatically* into a form which can be simulated. Let us analyze the difficulties by applying this model to the simple block on a rough surface shown in the right part of Figure 4 which is described by the equation:

$$m \cdot \dot{v} = u - f \tag{2}$$

where m is the mass of the block and $u(t)$ is the given driving force. If the element is in its *forward sliding* mode, i.e., $s \geq 1$, this model is described by

$$\begin{aligned} m \cdot \dot{v} &= u - f \\ v &= s - 1 \\ f &= f_0 + f_1 \cdot (s-1) \end{aligned}$$

which can be easily transformed into state space form with v as the state. If the block becomes stuck, i.e., $-1 \leq s \leq 1$, the equation $v = 0$ becomes active and therefore v can no longer be a state, i.e., an DAE index change takes place. Besides the difficulty to handle the DAE index change, there is a more serious problem: Assume that the block is stuck and that s becomes greater than one. Before the event occurs, $s \leq 1$ and $v = 0$; at the event instant $s > 1$ because this relation is the event triggering condition. The element switches into the forward sliding mode where v is a state which is initialized with its last value $v = 0$. Since v is a state, s is computed from v via $s := v+1$, resulting in $s = 1$, i.e., the relation $s > 1$ becomes **false** and the element switches back into the stuck mode. In other words, it is never possible to switch into the forward sliding mode. Taking numerical errors into account, the situation is even worse.

The key to the solution is the observation that $v = 0$ in the stuck mode and when forward sliding starts, but $\dot{v} > 0$ when sliding starts and $\dot{v} = 0$ in the stuck mode, see Figure 5. Since the friction characteristic in Figure 5 for $v = 0$ is no functional relationship, a parameterized curve description with a new curve parameter s_a leads for $v = 0$ to the equations

```
startFor  = sa >  1; startBack = sa < -1;
a = der(v);
a = if startFor  then  sa-1 else
    if startBack then  sa+1 else 0;
f = if startFor  then  f0   else
    if startBack then -f0   else f0*sa;
```

At zero velocity, these equations and the equation of the block (2) form a mixed continuous/discrete set of equations which has to be solved at event instants. The velocity v is kept as a state in all switching configurations. When switching from sliding to stuck

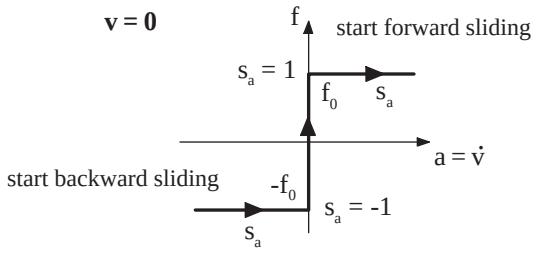
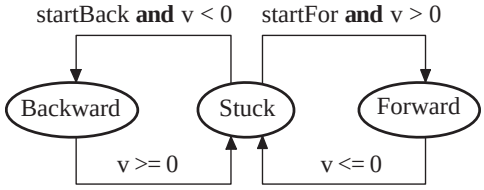


Figure 5: Friction characteristic at $v = 0$.

mode, the velocity is small or zero. Since the derivative of the constraint equation $\dot{v} = 0$ is fulfilled in the stuck mode, the velocity remains small even if $v = 0$ is not explicitly taken into account. Thus, v is small but may have any sign when switching from stuck to sliding mode; if the friction element starts to slide, say in the forward direction, one has to wait until the velocity is really positive, before switching to forward mode. (Even for exact calculation without numerical errors a "waiting" phase is necessary, because $v = 0$ when sliding starts.) Since $\dot{v} > 0$, this will occur after a small time period. This "waiting" procedure is described by the state machine



Putting the pieces together gives

```
// part of mixed system of equations
startFor  = pre(mode)==Stuck and sa >  1;
startBack = pre(mode)==Stuck and sa < -1;
a = der(v);
if pre(mode)==Forward or startFor
  then a = sa-1; f =  f0 + f1*v;
else if pre(mode)==Backward or startBack
  then a = sa+1; f = -f0 + f1*v;
else   a = 0;    f =  f0*sa;
end if;
// state machine to determine configuration
mode=
 if (pre(mode)==Forward or startFor) and v>0
   then Forward
 else if (pre(mode)==Backward or startBack)
                                    and v<0
   then Backward
 else Stuck;
```

The equations in the mixed system are evaluated based on the value of "mode" when the event occurred, i.e., on **pre**(mode). After the new sliding or stuck mode is determined by the solution of a mixed set of continuous/discrete equations, the new value of mode is computed by the last equation which is just a direct mapping of the state machine.

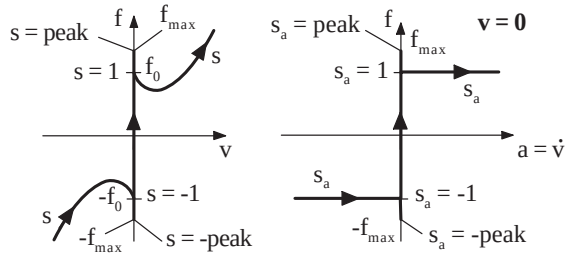


Figure 6: Coulomb friction characteristics.

It is now easy to model more general friction elements as in Figure 6 where the sliding friction force has a nonlinear characteristic and jumps from f_{max} to f_0 when sliding starts. The element equations of the simple friction element need only two changes: (1) the linear equation of the sliding friction force has to be replaced by an appropriate relationship and (2) using peak $= f_{max}/f_0 \geq 1$, the sliding conditions have to be modified to

```
startFor  = pre(mode)==Stuck and
   (sa>peak  or pre(startFor)  and sa>1);
startBack = pre(mode)==Stuck and
   (sa<-peak or pre(startBack) and sa<-1);
```

Real-Time Simulation

Let us consider real-time simulation of automatic gearboxes to illustrate what can be achieved. Figure 7 shows a Modelica model of the automatic gearbox ZF4HP22 as a composition diagram. It is set up by three standard planetary wheelsets, by shafts including wheel inertias, by clutches and by combined clutch/freewheel elements. The modeling of the gearbox is discussed further in [10].

Control is performed mostly by an electronic control unit, ECU, which generates the gearshift signals. Fine tuning of the ECU and the switching elements is essential to optimize gearshift comfort. In order to speed up the development cycle, it is natural to use simulation. However, the ECU is a proprietary component. The vendors do not provide any ECU models. Hardware-in-the-loop, HIL, simulation must be used, where the setup consists of the ECU hardware and a real-time simulation of all other components interacting dynamically with it. This requires very efficient simulation. A typical gearbox ECU sampling time is 10 ms implying that the simulation needs to produce values each 1 ms or faster.

The resulting mathematical model is a mixed system of Boolean equations and differential-algebraic equations with hundreds of unknown variables. There are no general-purpose solvers for such problems. There are numerical DAE solvers, which could be used to solve the continuous part, but they are at least 100 times too slow. The traditional approach has been to manually manipulate the model equations for each mode of operation and exploit special features to generate efficient code for numerical solution. Dymola automates all this work and generates efficient code.

The model in Figure 7 has five clutches (C4, C5, C6, C8, C11 and C12). Below we report results from a more complex model having 11 clutches. It is a gearbox model used by a major automotive company. The execution times are given for a 500 MHz DEC alpha processor from dSPACE. Please, note that this is not an extreme computing power today. It is actually available in modern PCs.

Dymola's BLT partitioning procedure finds that the gearbox model with 11 clutches has an algebraic loop including 212 unknowns. However, many of the involved are trivial connection equations of the form $v_1 = v_2$, which are easy to exploit for elimination giving an equation system with 55 unknowns. Tearing reduces the size of the equation system that has to be solved numerically to 23 unknowns, which is a considerable reduction. However, it is not enough for real-time simulation. Using Euler forward integration method and LU-factorization to solve the linear systems of equations, this simulation code needs more than 2 ms to produce output values. The numerical solution of the equation system takes is the major part to produce new values, because Euler forward method is very simple integration scheme. The situation can be improved a bit by fixing all parameter values at translation time. Dymola can then reduce the size of the equation system that has to be solved numerically to 14 unknowns, but the execution time needed is still too long.

The gearbox model has varying structure depending on whether the clutches are sliding and not. This fact can be utilized and special code generated for each individual mode. The model in Figure 7 has five switching elements (C4, C5, C6, C8, C11 and C12), which means that there are $2^6 = 64$ possible configurations of the systems. It is indeed possible to treat all the 64 cases individually. However, the gearbox discussed with 11 clutches has $2^{11} = 2048$ possible configurations. Fortunately, it turns out that for typical drive cycles only 10 to 30 modes are active.

Dymola includes a facility to find out which variables shall be considered to define a mode. Off-line simulations are run to find out which configurations are active, Dymola logs and outputs a list of the active modes after the simulation. The list is fed back to Dymola to be used in the next translation of the model. When the translation procedure has done the tearing and as a result have obtained the equation system of size 14, the translator generates special code for each used combination of clutch locking combina-

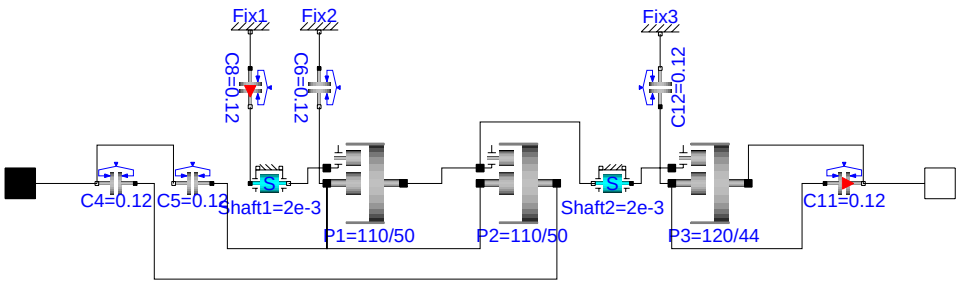


Figure 7: Composition diagram of gearbox ZF4HP22.

tions and for the general case as well. For each mode value it substitutes the values of the mode variables and simplifies the equations. Then the equation system is BLT partitioned. For some mode values the result is a simple sequence of scalar problems, which are easy to solve symbolically since the problem in this case is linear. For other values there remain equation systems of size 9. For real-time simulations these "worst cases" set the limit for the sampling rate. However, there are possibilities to precalculation. The substitution of the mode values makes the matrix J time invariant; the elements only depends on physical parameters (moment of inertias and gear ratios), making it is possible to precalculate J and its LU-factorization.

Dymola performs all the manipulations described above very fast. It takes Dymola only a few seconds to translate the Modelica model of the gearbox with 11 switching elements into efficient simulation code. A 500 MHz DEC alpha processor from dSPACE evaluates one Euler step including a possible mode switch in less than 0.18 ms.

Conclusions

Modeling of variable structure models in Modelica has been discussed and illustrated for ideal diodes and Coulomb friction. Approaches to efficient simulations have been outlined and the power of symbolic manipulation as supported by Dymola has been demonstrated for a real industrial application.

Acknowledgements

The work has been supported by the Swedish National Board for Technical Development (NUTEK project 1K1P-97-09759). The authors would like to thank Hans Olsson, Dynasim for his contribution to the development and the implementation of Dymola's numerical solver for mixed equation systems.

References

[1] C. Clauß, J. Haase, G. Kurth, and P. Schwartz. "Extended admittance description of nonlinear n-poles." *Archiv für Elektronik und Übertragungstechnik / International Journal Of Electronics and Communications*, **40**, pp. 91–97, 1995.

[2] Dymola. *http://www.dynasim.se/.* Dynasim AB.

[3] H. Elmqvist. "An object and data-flow based visual language for process control." In *ISA/92-Canada Conference & Exhibit*, Toronto, Canada, April 1992. Instrument Society of America.

[4] H. Elmqvist, B. Bachmann, F. Boudaud, J. Broenink, D. Brück, T. Ernst, R. Franke, P. Fritzson, A. Jeandel, P. Grozman, K. Juslin, D. Kågedal, M. Klose, N. Loubere, S. Mattsson, P. Mosterman, H. Nilsson, M. Otter, P. Sahlin, A. Schneider, H. Tummescheit, and H. Vangheluwe. *Modelica^{TM} — A Unified Object-Oriented Language for Physical Systems Modeling, (Tutorial and Rationale), Version 1.2.* http://www.modelica.org, 1999.

[5] H. Elmqvist, F. Cellier, and M. Otter. "Object-oriented modeling of hybrid systems." In *Proceedings of European Simulation Symposium, ESS'93*. The Society of Computer Simulation, October 1993.

[6] H. Elmqvist, S. E. Mattsson, and M. Otter. "Modelica — A language for physical system modeling, visualization and interaction." In *Proceedings of the 1999 IEEE Symposium on Computer-Aided Control System Design, CACSD'99*, Hawaii, USA, August 1999. Plenary talk.

[7] T. Gautier, P. Le Guernic, and O. Maffeis. "For a new real-time methodology." Publication Interne No 870, Institut de Recherche en Informatique et Systemes Aleatoires, Campus de Beaulieu, Rennes, France, 1994.

[8] N. Halbwachs. *Synchronous Programming of Reactive Systems*. Kluwer, 1993.

[9] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. "The synchronous data flow programming language LUSTRE." *Proc. of the IEEE*, **79**, pp. 1305–1321, 1991.

[10] S. E. Mattsson, H. Elmqvist, and M. Otter. "Physical system modeling with Modelica." *Control Engineering Practice*, **6**, pp. 501–510, 1998.

[11] M. Otter, H. Elmqvist, and S. E. Mattsson. "Hybrid modeling in Modelica based on the synchronous data flow principle." In *Proceedings of the 1999 IEEE Symposium on Computer-Aided Control System Design, CACSD'99*, Hawaii, USA, August 1999.